Project: Stock Market Analysis

Introduction

The Nasdaq-100 is a stock market index comprised of 102 equity securities issued by 101 of the Nasdaq's largest nonfinancial companies. It includes sectors such as manufacturing, technology, retail, telecommunication, biotechnology, health care, transportation, media, and service providers.
The cluster trading strategy is used to build a diverse portfolio of investments. This method enables the identification of different company segments. One advantage of this analysis is that it can help to protect an investor's portfolio from risks.

Objective

You must now create such segments so that customers can identify segments to invest in and segments to avoid. Use cluster analysis techniques to accomplish this task. You will also need to perform time-series forecasting for stock prices.

Submission

```
# For Data Processing
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader import DataReader

# For time stamps
from datetime import datetime

# For division
from __future__ import division
```

```python
# List of Tech_stocks for analytics
tech_list = ['AAPL','GOOGL','MSFT','AMZN']

# set up Start and End time for data grab
end = datetime.now()
start = datetime(end.year-1,end.month,end.day)

#For-loop for grabing finance data and setting as a dataframe
# Set DataFrame as the Stock Ticker

for stock in tech_list:
    globals()[stock] = DataReader(stock,'google',start,end)
```

```python
AAPL.head()
```

	Open	High	Low	Close	Volume
Date					
2016-10-27	115.39	115.86	114.10	114.48	34562045
2016-10-28	113.87	115.21	113.45	113.72	37861662
2016-10-31	113.65	114.23	113.20	113.54	26419398
2016-11-01	113.46	113.77	110.53	111.49	43825812
2016-11-02	111.40	112.35	111.23	111.59	28331709

```python
# Summery stats for Apple Stock
AAPL.describe()
```

	Open	High	Low	Close	Volume
count	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	139.381474	140.304622	138.494263	139.488327	2.805794e+07
std	17.106701	17.101638	16.891555	16.951448	1.193381e+07
min	106.570000	107.680000	104.080000	105.710000	1.147592e+07
25%	120.435000	121.100000	120.025000	120.715000	2.082378e+07
50%	143.720000	144.500000	143.100000	143.700000	2.559729e+07
75%	153.880000	154.450000	152.900000	153.805000	3.195864e+07
max	164.800000	164.940000	163.630000	164.050000	1.119850e+08

```python
# General Info
AAPL.info()
```

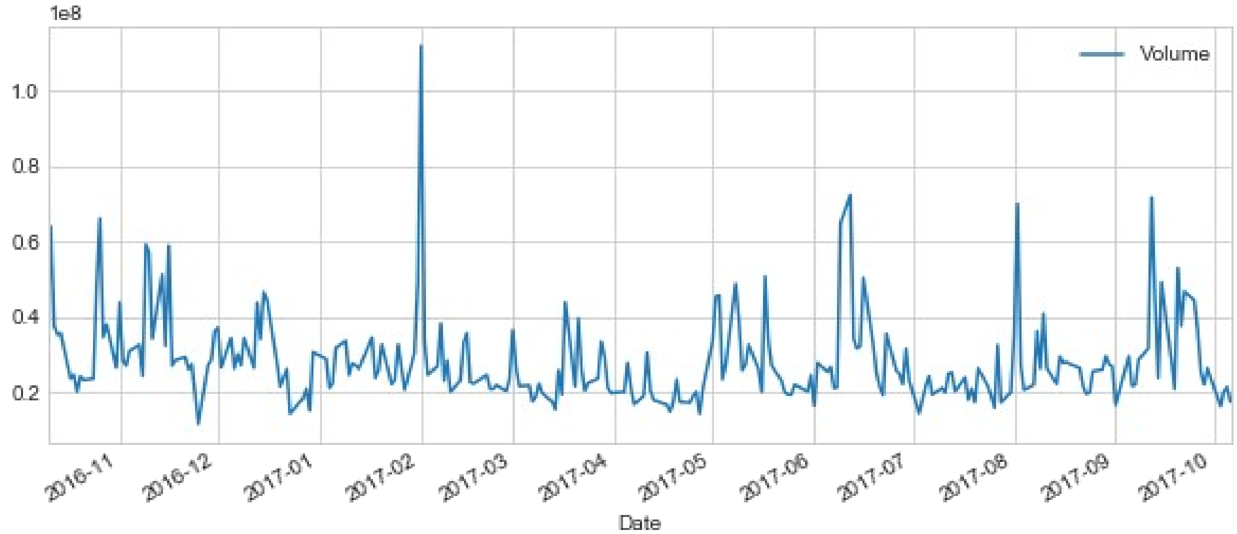
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2016-10-10 to 2017-10-06
Data columns (total 5 columns):
```

```
Open       251 non-null float64
High       251 non-null float64
Low        251 non-null float64
Close      251 non-null float64
Volume     251 non-null int64
dtypes: float64(4), int64(1)
memory usage: 11.8 KB
```

Now that we've seen the DataFrame, let's go ahead and plot out the volume and closing price of the AAPL(Apple) stocks.

```
# Let's see a historical view of the closing price
AAPL['Close'].plot(legend=True, figsize=(10,4))

<matplotlib.axes._subplots.AxesSubplot at 0x23b914cf9b0>
```


```
# Now let's plot the total volume of stock being traded each day over the past year

AAPL['Volume'].plot(legend=True, figsize=(10,4))

<matplotlib.axes._subplots.AxesSubplot at 0x23b90cf27f0>
```




```
MA_day = [10,20,50,100]

for ma in MA_day:
    column_name = 'MA for %s days' %(str(ma))
    AAPL[column_name] = pd.rolling_mean(AAPL['Close'],ma)
```

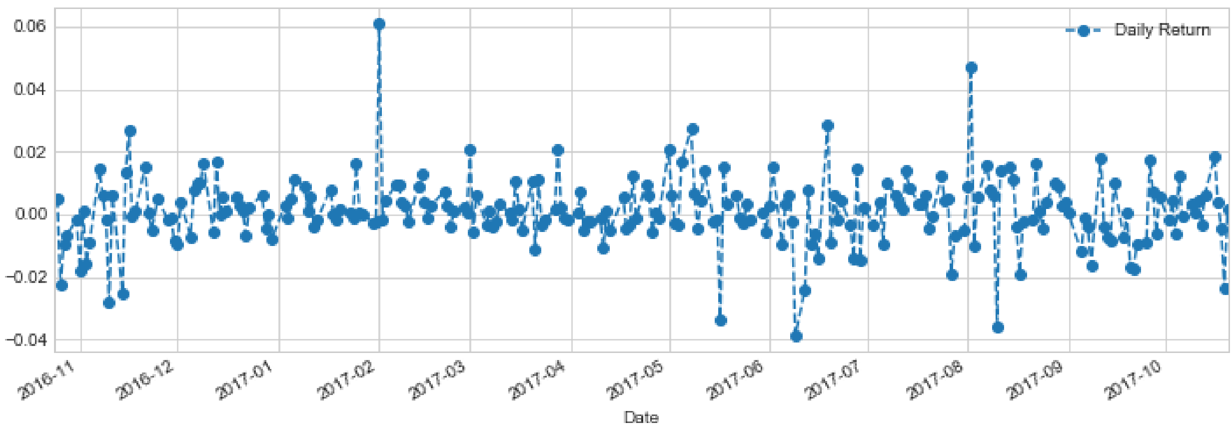
```
c:\users\admin\anaconda2\envs\python3.5\lib\site-packages\
ipykernel_launcher.py:8: FutureWarning: pd.rolling_mean is deprecated
for Series and will be removed in a future version, replace with
      Series.rolling(center=False,window=10).mean()

c:\users\admin\anaconda2\envs\python3.5\lib\site-packages\
ipykernel_launcher.py:8: FutureWarning: pd.rolling_mean is deprecated
for Series and will be removed in a future version, replace with
      Series.rolling(center=False,window=20).mean()

c:\users\admin\anaconda2\envs\python3.5\lib\site-packages\
ipykernel_launcher.py:8: FutureWarning: pd.rolling_mean is deprecated
for Series and will be removed in a future version, replace with
```

```
      Series.rolling(center=False,window=50).mean()

c:\users\admin\anaconda2\envs\python3.5\lib\site-packages\
ipykernel_launcher.py:8: FutureWarning: pd.rolling_mean is deprecated
for Series and will be removed in a future version, replace with
      Series.rolling(center=False,window=100).mean()
```

Now, lets plot all the additional Moving Averages for AAPL stock

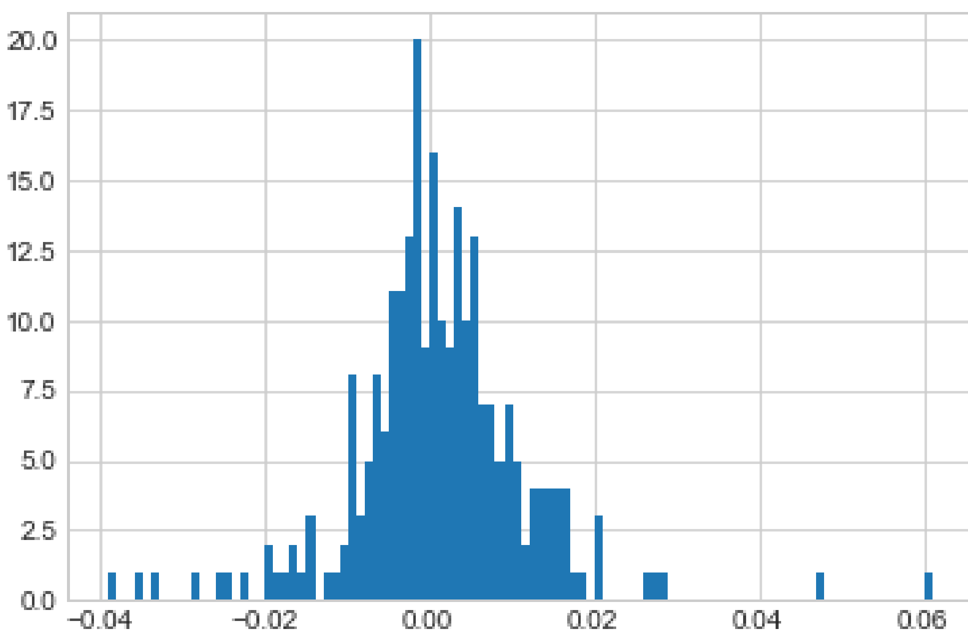
```
AAPL[['Close','MA for 10 days','MA for 20 days','MA for 50 days','MA
for 100 days']].plot(subplots=False,figsize=(10,4))

<matplotlib.axes._subplots.AxesSubplot at 0x23b90900eb8>
```


```
# We'll use pct_change to find the percent change for each day
AAPL['Daily Return'] = AAPL['Close'].pct_change()

# Lets plot the daily return percentage
AAPL['Daily Return'].plot(figsize=(12,4), legend=True, linestyle='--',
marker='o')

<matplotlib.axes._subplots.AxesSubplot at 0x12991f54278>
```

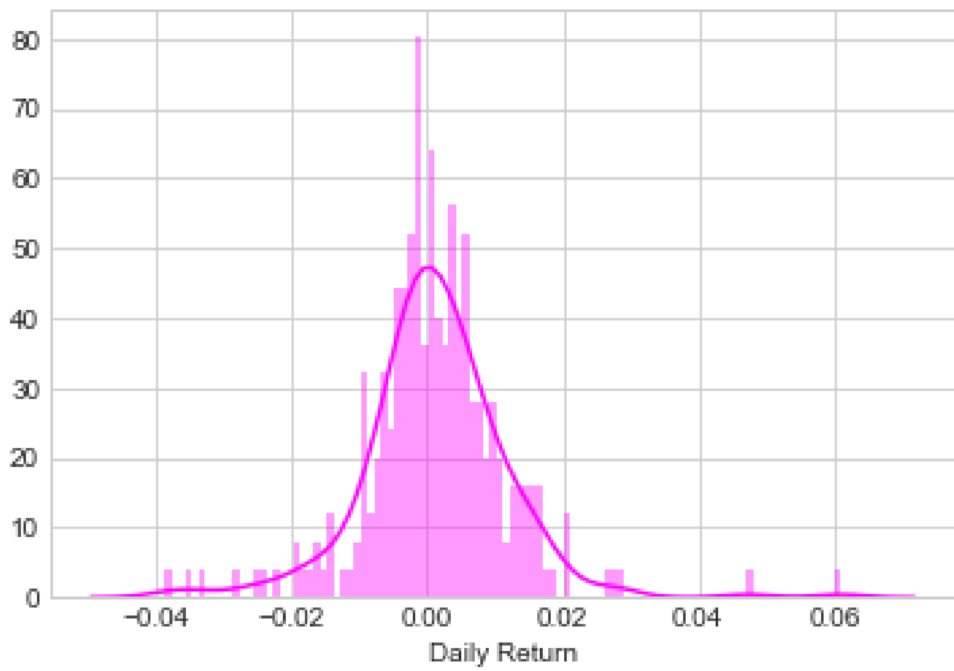


Great, now let's get an overall look at the average daily return using a histogram. By using seaborn to create both a histogram and kde plot on the same figure.

```
# only with histogram
AAPL['Daily Return'].hist(bins=100)

<matplotlib.axes._subplots.AxesSubplot at 0x1a6475edf60>
```




```
# Note the use of dropna() here, otherwise the NaN values can't be
read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(), bins=100, color='magenta')

<matplotlib.axes._subplots.AxesSubplot at 0x1a64764e7f0>
```



Now what if we wanted to analyze the returns of all the stocks in our list? For that, we need to build a DataFrame with all the ['Close'] columns for each of the stocks dataframes.

```
# Grab all the closing prices for the tech stock list into one
DataFrame

closingprice_df = DataReader(tech_list, 'google', start, end)['Close']

closingprice_df.head(10)
```

	AAPL	AMZN	GOOGL	MSFT
Date				
2016-10-17	117.55	812.95	806.84	57.22
2016-10-18	117.47	817.65	821.49	57.66
2016-10-19	117.12	817.69	827.09	57.53
2016-10-20	117.06	810.32	821.63	57.25
2016-10-21	116.60	818.99	824.06	59.66
2016-10-24	117.65	838.09	835.74	61.00
2016-10-25	118.25	835.18	828.55	60.99
2016-10-26	115.59	822.59	822.10	60.63
2016-10-27	114.48	818.36	817.35	60.10
2016-10-28	113.72	776.32	819.56	59.87

Now that we have all the closing prices, let's go ahead and get the daily return for all the stocks, like we did for the APPL stock.

```
# make a new tech returns DataFrame
tech_returns = closingprice_df.pct_change()
```

```
tech_returns.head()
```

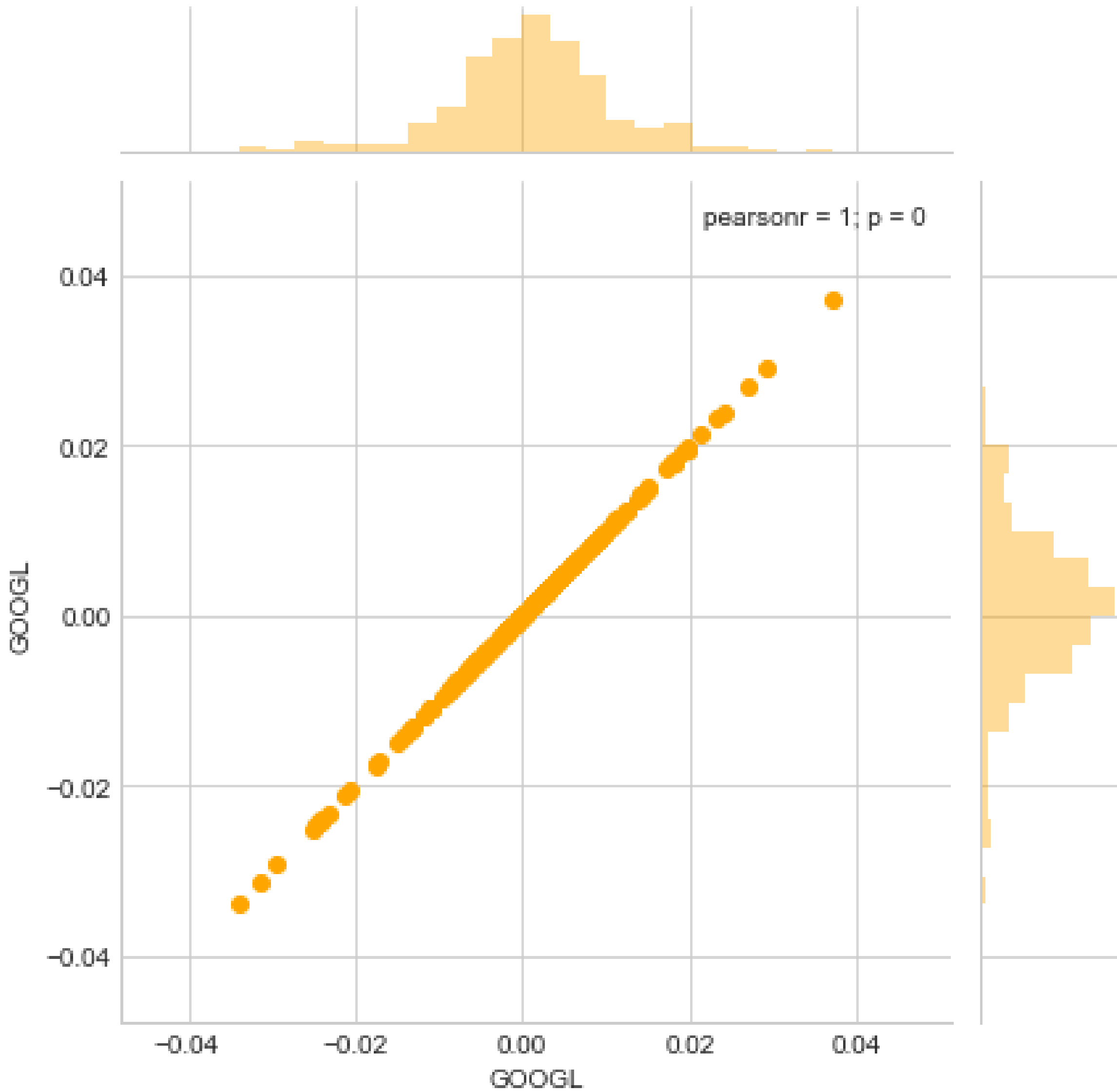
```
                AAPL      AMZN     GOOGL      MSFT
Date
2016-10-24       NaN       NaN       NaN       NaN
2016-10-25  0.005100 -0.003472 -0.008603 -0.000164
2016-10-26 -0.022495 -0.015075 -0.007785 -0.005903
2016-10-27 -0.009603 -0.005142 -0.005778 -0.008742
2016-10-28 -0.006639 -0.051371  0.002704 -0.003827
```

Now we can compare the daily percentage return of two stocks to check how correlated. First let's see a stock compared to itself.

GOOGL is a Alphabet Inc Class A Stock.

```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot('GOOGL','GOOGL',tech_returns,kind='scatter',color='orange')
```

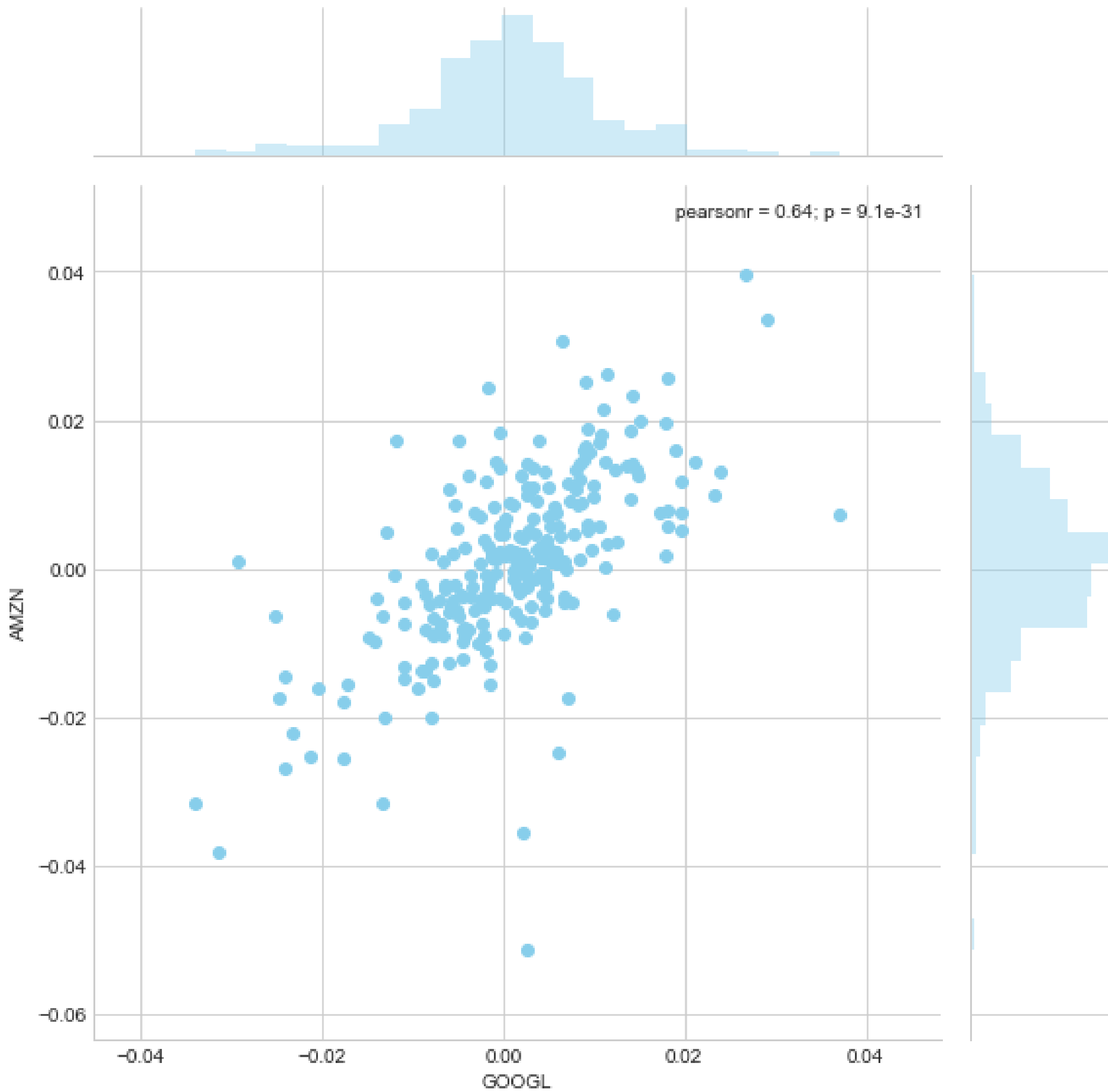
```
<seaborn.axisgrid.JointGrid at 0x1a647cc71d0>
```



So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

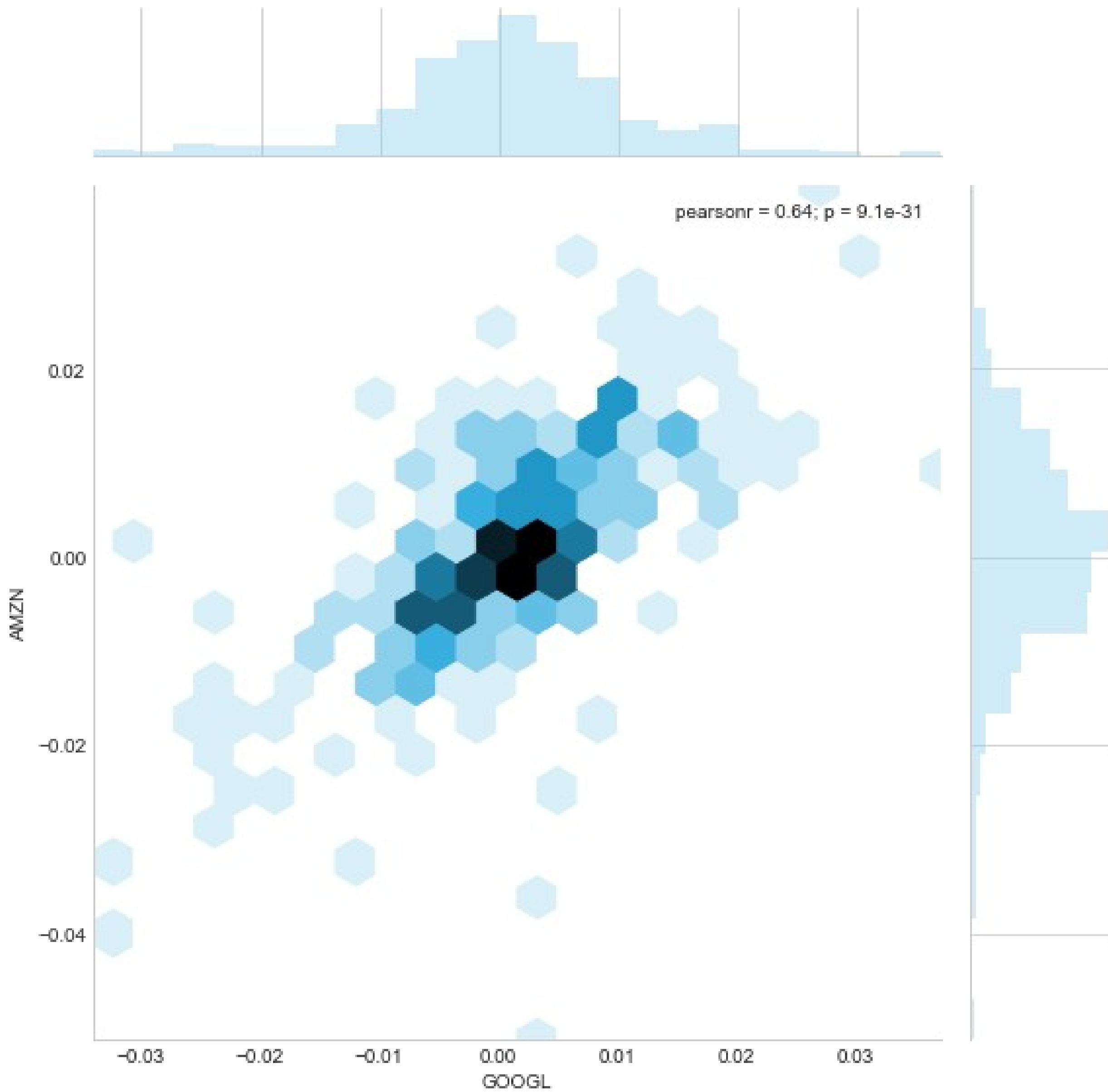
So let's go ahead and compare Google and Amazon the same way.

```
# We'll use joinplot to compare the daily returns of Google and
Amazon.

sns.jointplot('GOOGL','AMZN',tech_returns, kind='scatter',size=8,
color='skyblue')

<seaborn.axisgrid.JointGrid at 0x1a648484d68>
```

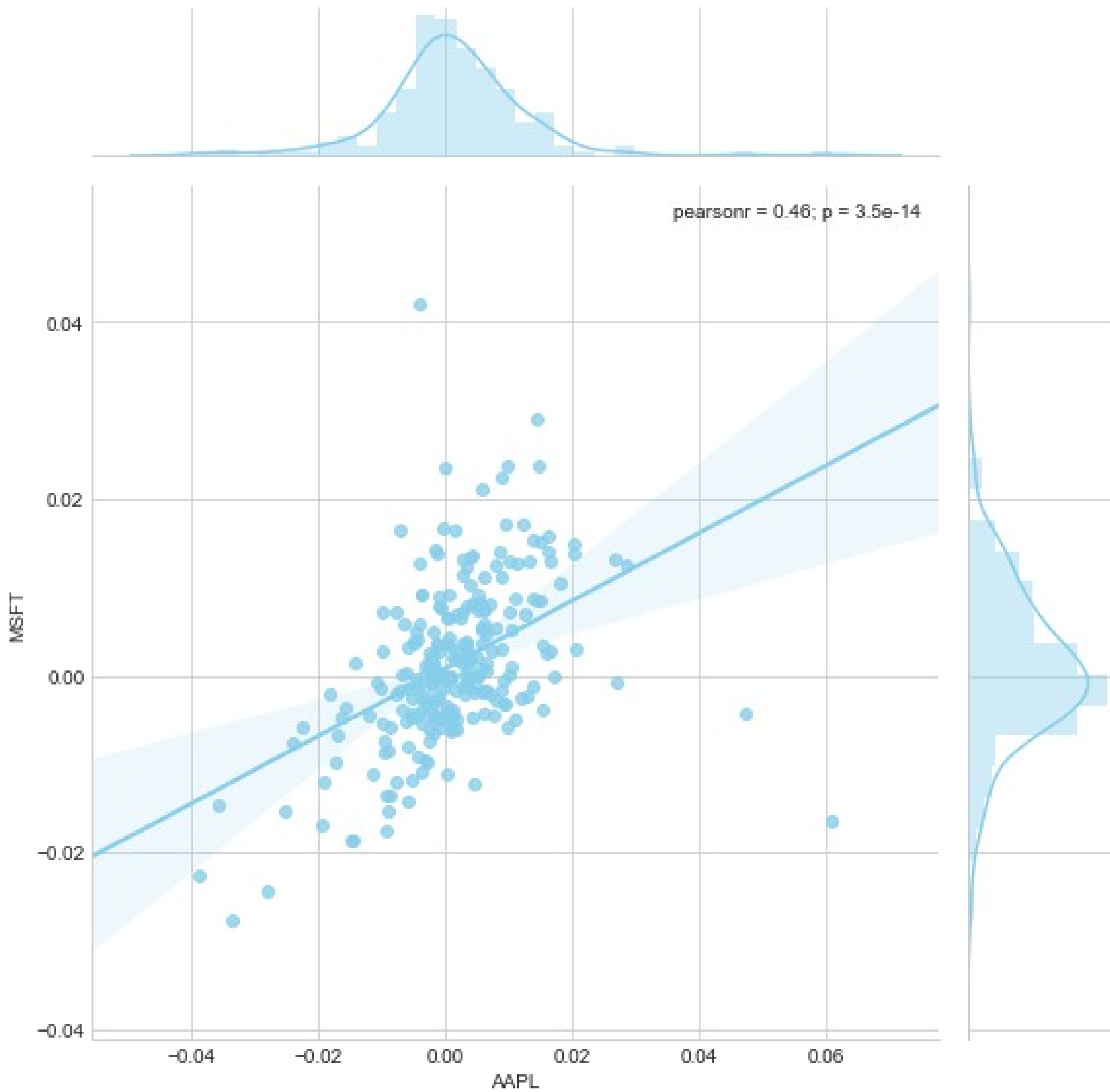



```
# with Hex plot
sns.jointplot('GOOGL','AMZN',tech_returns, kind='hex',size=8,
color='skyblue')
```

```
<seaborn.axisgrid.JointGrid at 0x1a649727a90>
```




```
# Lets check out for Apple and Microsoft with reg jointplot
sns.jointplot('AAPL','MSFT',tech_returns, kind='reg', size=8,
color='skyblue')

<seaborn.axisgrid.JointGrid at 0x1a64a4e8198>
```



Intersting, the pearsonr value (officially known as the Pearson product-moment correlation coefficient) can give you a sense of how correlated the daily percentage returns are. You can find more information about it at this link:

Url - http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient

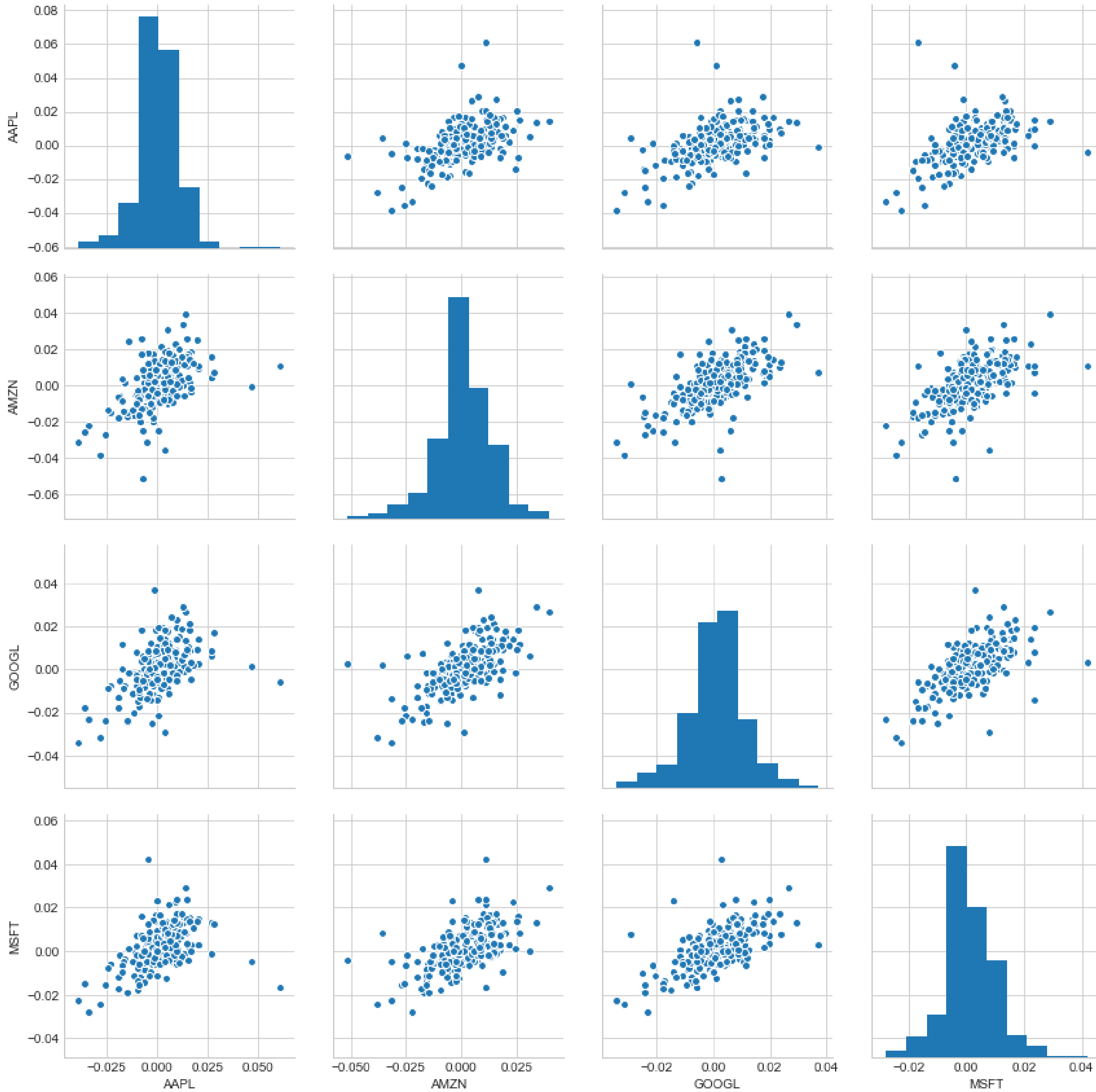
But for a quick intuitive sense, check out the picture below.

```
from IPython.display import SVG
SVG(url='http://upload.wikimedia.org/wikipedia/commons/d/d4/Correlatio
n_examples2.svg')
```

Seaborn and Pandas make it very easy to repeat this comparison analysis for every possible combination of stocks in our technology stock ticker list. We can use sns.pairplot() to automatically create this plot

```
# We can simply call pairplot on our DataFrame for an automatic visual
analysis of all the comparisons
sns.pairplot(tech_returns.dropna(),size=3)
```

```
<seaborn.axisgrid.PairGrid at 0x1a665444ba8>
```



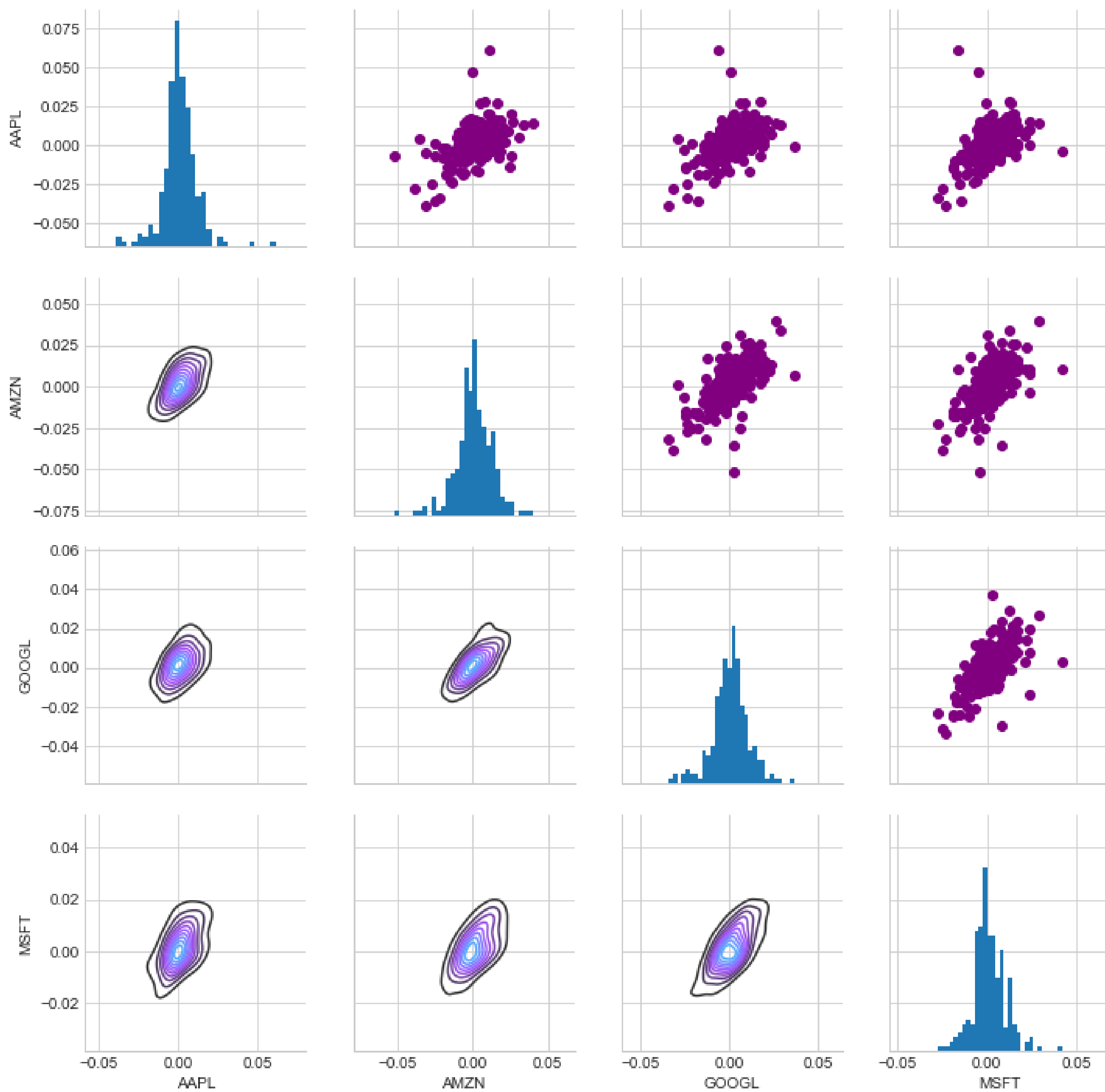
Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison. While the simplicity of just calling sns.pairplot() is fantastic we can also use sns.PairGrid() for full control of the figure, including what kind of plots go in the diagonal, the upper triangle, and the lower triangle.

Below is an example of utilizing the full power of seaborn to achieve this result.

```
# Set up the figure by naming it returns_fig, call PairGrid on the
DataFrame
returns_fig = sns.PairGrid(tech_returns.dropna())

# Using map_upper we can specify what the upper triangle will look
```

```
like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, including the
plot type (kde) & the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of
the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

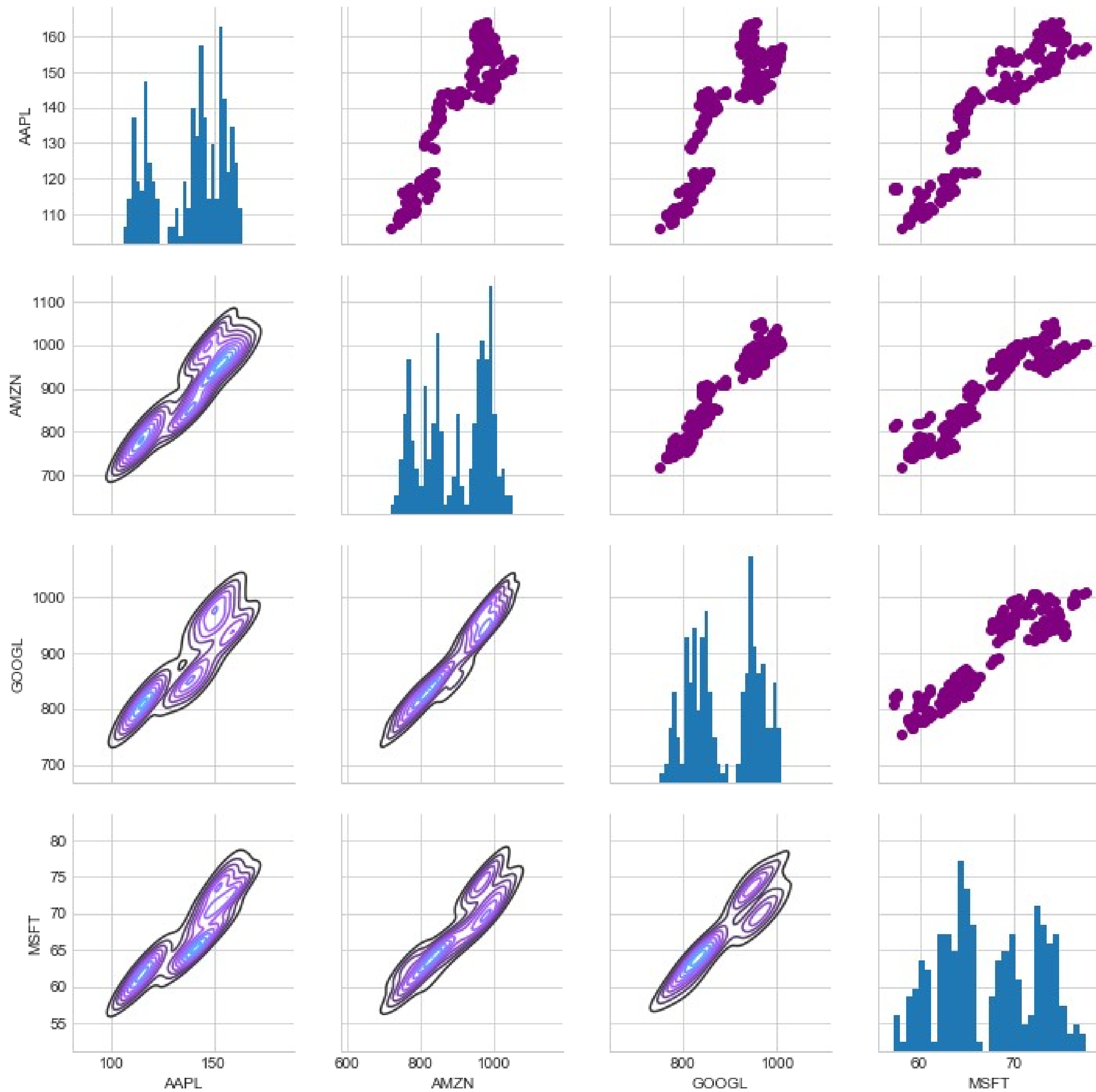
```
<seaborn.axisgrid.PairGrid at 0x1a6670f82b0>
```



We can also analyze the correlation of the closing prices using this exact same technique. Here it is shown, the code repeated from above with the exception of the DataFrame called.

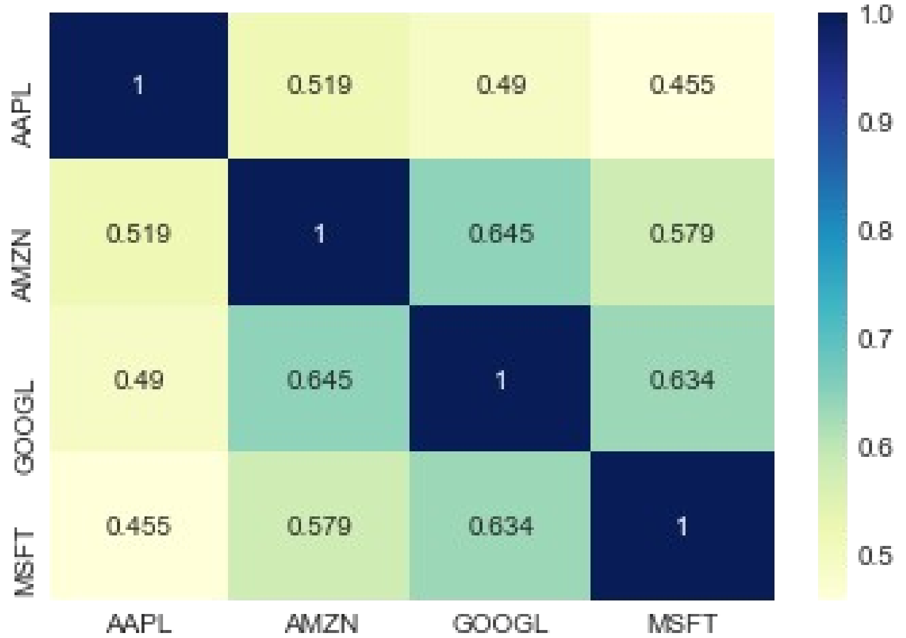
```
# Set up the figure by naming it returns_fig, call PairGrid on the
DataFrame
returns_fig = sns.PairGrid(closingprice_df.dropna())

# Using map_upper we can specify what the upper triangle will look
like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, including the
plot type (kde) & the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of
the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

```
<seaborn.axisgrid.PairGrid at 0x1a666b2c3c8>
```



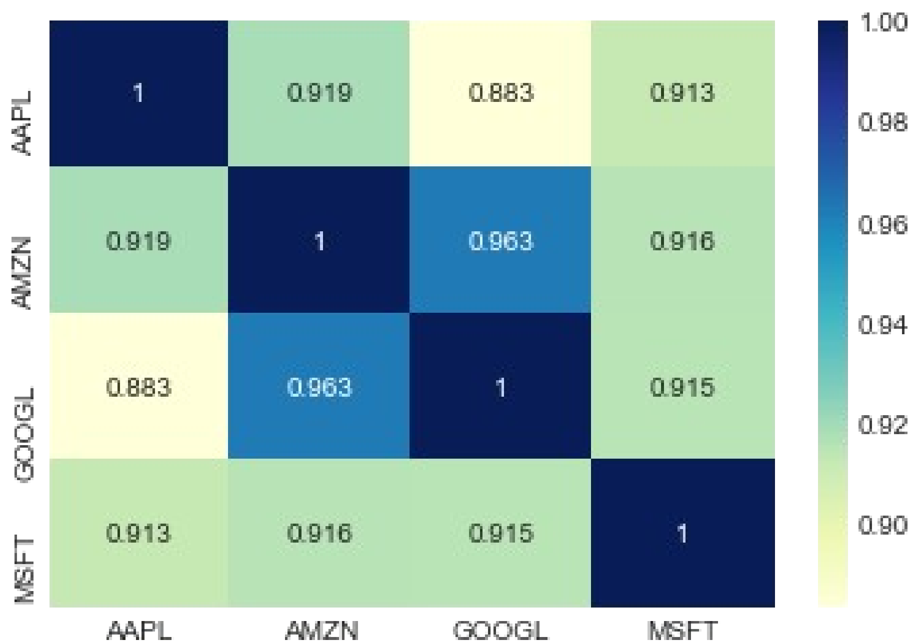
Finally, we can also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Google and Amazon stocks.

```
# Let's go ahead and use seaborn for a quick heatmap to get
correlation for the daily return of the stocks.
sns.heatmap(tech_returns.corr(),annot=True,fmt=".3g",cmap='YlGnBu')

<matplotlib.axes._subplots.AxesSubplot at 0x1a67363a780>
```




```python
# Lets check out the correlation between closing prices of stocks
sns.heatmap(closingprice_df.corr(),annot=True,fmt=".3g",cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a6736767b8>
```



Fantastic! Just like we suspected in our PairPlot we see here numerically and visually that Amazon and Google had the strongest correlation of daily stock return. It's also interesting to see that all the technology comapnies are positively correlated.

Great! Now that we've done some daily return analysis, let's go ahead and start looking deeper into actual risk analysis.

Risk Analysis

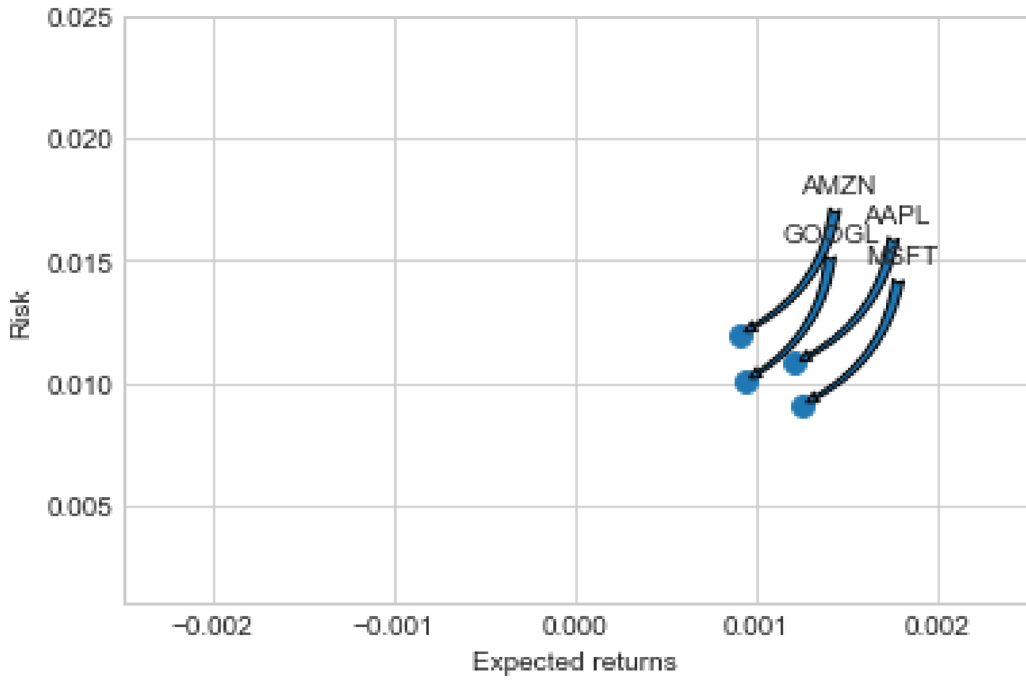
There are many ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns(Risk).

```
# Let's start by defining a new DataFrame as a clenaed version of the oriignal tech_returns DataFrame
rets = tech_returns.dropna()

rets.head()
```

	AAPL	AMZN	GOOGL	MSFT
Date				
2016-10-25	0.005100	-0.003472	-0.008603	-0.000164
2016-10-26	-0.022495	-0.015075	-0.007785	-0.005903
2016-10-27	-0.009603	-0.005142	-0.005778	-0.008742
2016-10-28	-0.006639	-0.051371	0.002704	-0.003827
2016-10-31	-0.001583	0.017390	-0.011787	0.000835

```
# Defining the area for the circles of scatter plot to avoid tiny little points
area = np.pi*20

plt.scatter(rets.mean(),rets.std(),s=area)

# Set the x and y limits of the plot (optional, remove this if you don't see anything in your plot)
plt.xlim([-0.0025,0.0025])
plt.ylim([0.001,0.025])

#Set the plot axis titles
plt.xlabel('Expected returns')
plt.ylabel('Risk')

# Label the scatter plots, for more info on how this is done, chekc out the link below
# http://matplotlib.org/users/annotations_guide.html
for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(
        label,
        xy = (x, y), xytext = (50, 50),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
```

```
        arrowprops = dict(arrowstyle = 'fancy', connectionstyle = 
'arc3,rad=-0.3'))
```



By looking at the scatter plot we can say these stocks have lower risk and positive expected returns.

Value at Risk

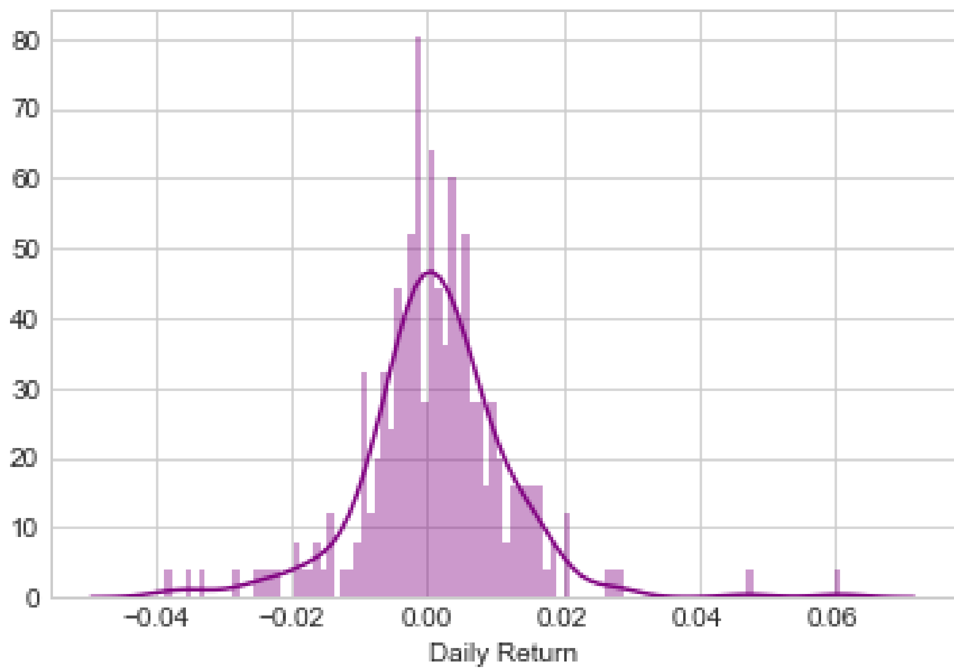
Let's go ahead and define a value at risk parameter for our stocks. We can treat value at risk as the amount of money we could expect to lose (aka putting at risk) for a given confidence interval. There's several methods we can use for estimating a value at risk. Let's go ahead and see some of them in action.

Value at risk using the "bootstrap" method

For this method we will calculate the empirical quantiles from a histogram of daily returns. For more information on quantiles, check out this link: http://en.wikipedia.org/wiki/Quantile

Let's go ahead and repeat the daily returns histogram for Apple stock.

```
# Note the use of dropna() here, otherwise the NaN values can't be 
read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(),bins=100,color='purple')

<matplotlib.axes._subplots.AxesSubplot at 0x12991fe7d30>
```



Now we can use quantile to get the risk value for the stock.

```
# The 0.05 empirical quantile of daily returns

# For APPL stocks
rets["AAPL"].quantile(0.05)
```

```
-0.01655598896390161
```

The 0.05 empirical quantile of daily returns is at -0.016. That means that with 95% confidence, our worst daily loss will not exceed 1.6%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.016 * 1,000,000 = $16,000.

```
# For AMZN stocks
rets["AMZN"].quantile(0.05)
```

```
-0.01774398557895971
```

```
# For GOOGL stocks
rets["GOOGL"].quantile(0.05)
```

```
-0.01614604826290949
```

```
# For MSFT stocks
rets["MSFT"].quantile(0.05)
```

```
-0.01356824956195934
```

Value at Risk using the Monte Carlo method

Using the Monte Carlo to run many trials with random market conditions, then we'll calculate portfolio losses for each trial. After this, we'll use the aggregation of all these simulations to establish how risky the stock is.

Let's start with a brief explanation of what we're going to do:

We will use the geometric Brownian motion (GBM), which is technically known as a Markov process. This means that the stock price follows a random walk and is consistent with (at the very least) the weak form of the efficient market hypothesis (EMH): past price information is already incorporated and the next price movement is "conditionally independent" of past price movements.

This means that the past information on the price of a stock is independent of where the stock price will be in the future, basically meaning, you can't perfectly predict the future solely based on the previous price of a stock.

Now we see that the change in the stock price is the current stock price multiplied by two terms. The first term is known as "drift", which is the average daily return multiplied by the change of time. The second term is known as "shock", for each time period the stock will "drift" and then experience a "shock" which will randomly push the stock price up or down. By simulating this series of steps of drift and shock thousands of times, we can begin to do a simulation of where we might expect the stock price to be.

For more info on the Monte Carlo method for stocks and simulating stock prices with GBM model ie. geometric Brownian motion (GBM).

check out the following link: http://www.investopedia.com/articles/07/montecarlo.asp

To demonstrate a basic Monte Carlo method, we will start with just a few simulations. First we'll define the variables we'll be using in the Google stock DataFrame GOOGL

```
rets.head()
```

Date	AAPL	AMZN	GOOGL	MSFT
2016-10-25	0.005100	-0.003472	-0.008603	-0.000164
2016-10-26	-0.022495	-0.015075	-0.007785	-0.005903
2016-10-27	-0.009603	-0.005142	-0.005778	-0.008742
2016-10-28	-0.006639	-0.051371	0.002704	-0.003827
2016-10-31	-0.001583	0.017390	-0.011787	0.000835

```
# Set up our time horizon
days = 365

# Now our delta
dt = 1/days

# Now let's grab our mu (drift) from the expected return data we got
for GOOGL
mu = rets.mean()['GOOGL']
```

```
# Now let's grab the volatility of the stock from the std() of the
average return for GOOGL
sigma = rets.std()['GOOGL']
```

Next, we will create a function that takes in the starting price and number of days, and uses the sigma and mu we already calculated form our daily returns.

```
def stock_monte_carlo(start_price,days,mu,sigma):
    ''' This function takes in starting stock price, days of
simulation,mu,sigma, and returns simulated price array'''

    # Define a price array
    price = np.zeros(days)
    price[0] = start_price

    # Schok and Drift
    shock = np.zeros(days)
    drift = np.zeros(days)

    # Run price array for number of days
    for x in range(1,days):

        # Calculate Schock
        shock[x] = np.random.normal(loc=mu * dt, scale=sigma *
np.sqrt(dt))
        # Calculate Drift
        drift[x] = mu * dt
        # Calculate Price
        price[x] = price[x-1] + (price[x-1] * (drift[x] + shock[x]))

    return price
```

Awesome! Now lets put above function to work.

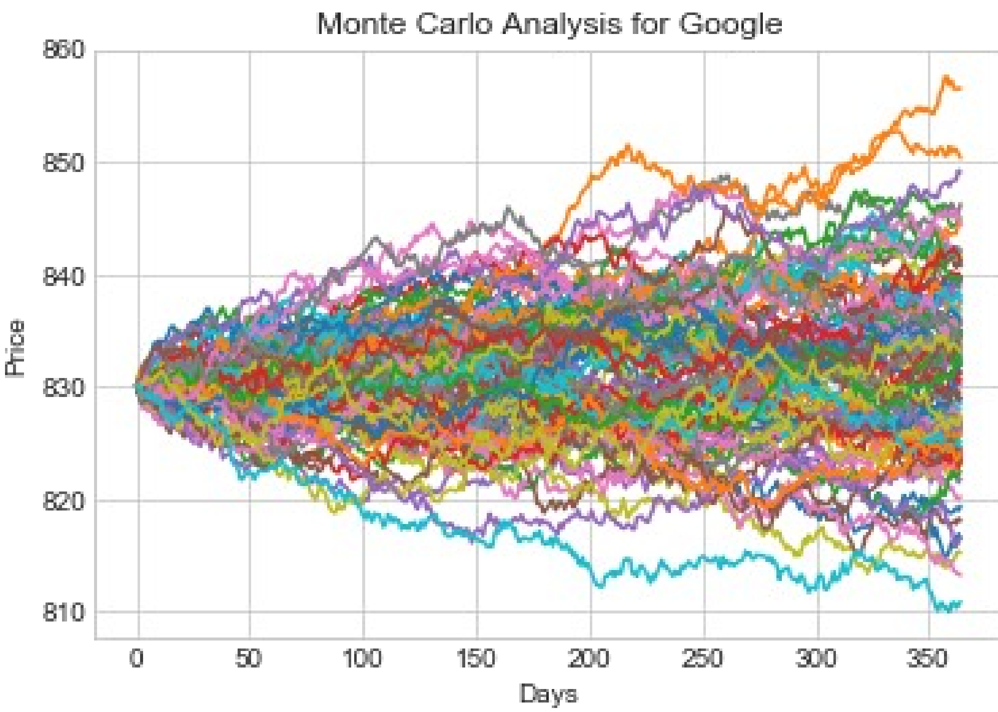
```
# For Google Stock - GOOGL
GOOGL.head()
```

	Open	High	Low	Close	Volume
Date					
2016-10-24	830.09	837.94	829.04	835.74	1447616
2016-10-25	838.50	838.50	825.30	828.55	1890712
2016-10-26	827.12	827.71	816.35	822.10	1794868
2016-10-27	823.01	826.58	814.61	817.35	2973486
2016-10-28	829.94	839.00	817.00	819.56	4354884

```
start_price = 830.09

for run in range(100):
```

```
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))

plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Google')
```

```
<matplotlib.text.Text at 0x12993b9e668>
```

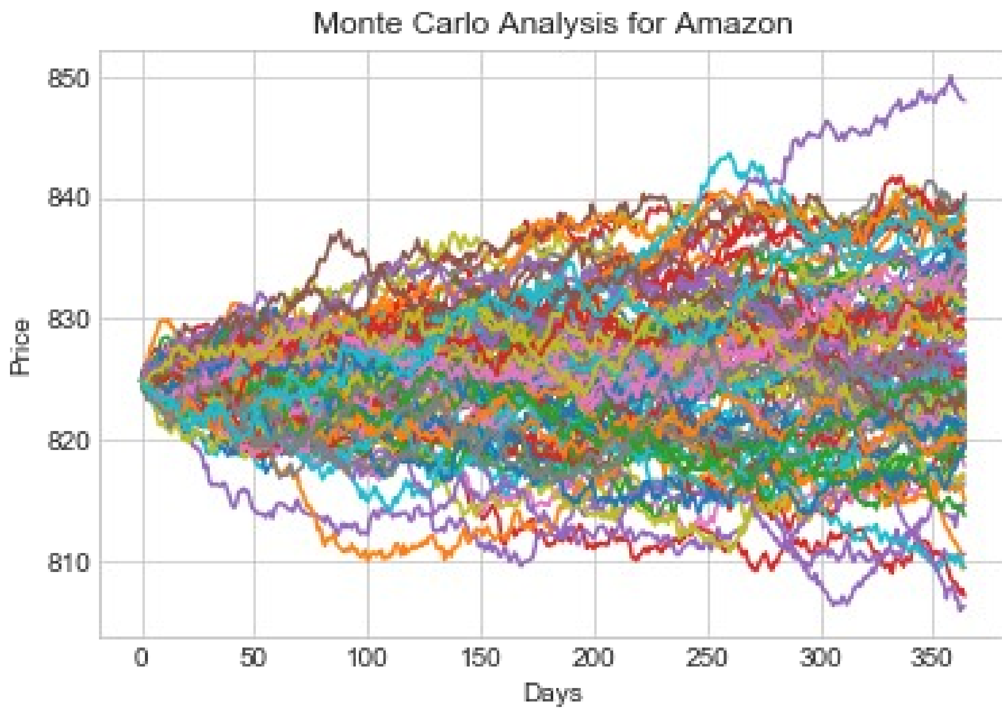



```
# For Amazon Stock - AMZN
AMZN.head()
```

	Open	High	Low	Close	Volume
Date					
2016-10-24	824.95	838.30	822.21	838.09	4060899
2016-10-25	839.30	843.09	833.22	835.18	3248358
2016-10-26	832.76	833.44	820.00	822.59	3998102
2016-10-27	831.24	831.72	815.43	818.36	7406385
2016-10-28	782.00	789.49	774.61	776.32	10841073

```
start_price = 824.95

for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))

plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Amazon')
```

```
<matplotlib.text.Text at 0x12993bdf160>
```

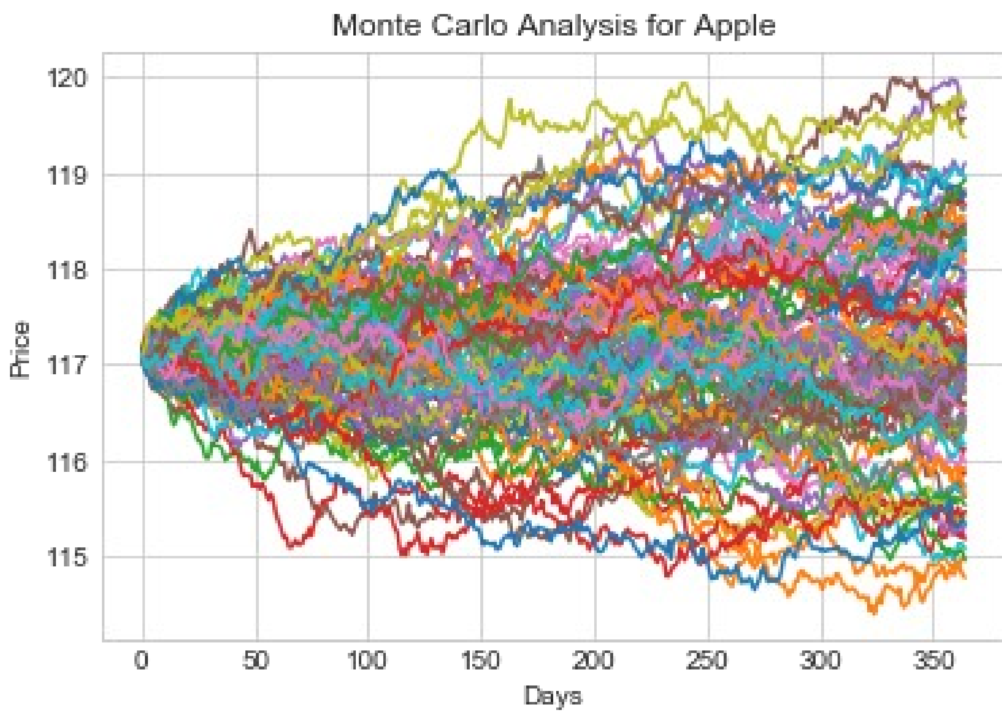



```
# For Apple Stock - AAPL
AAPL.head()
```

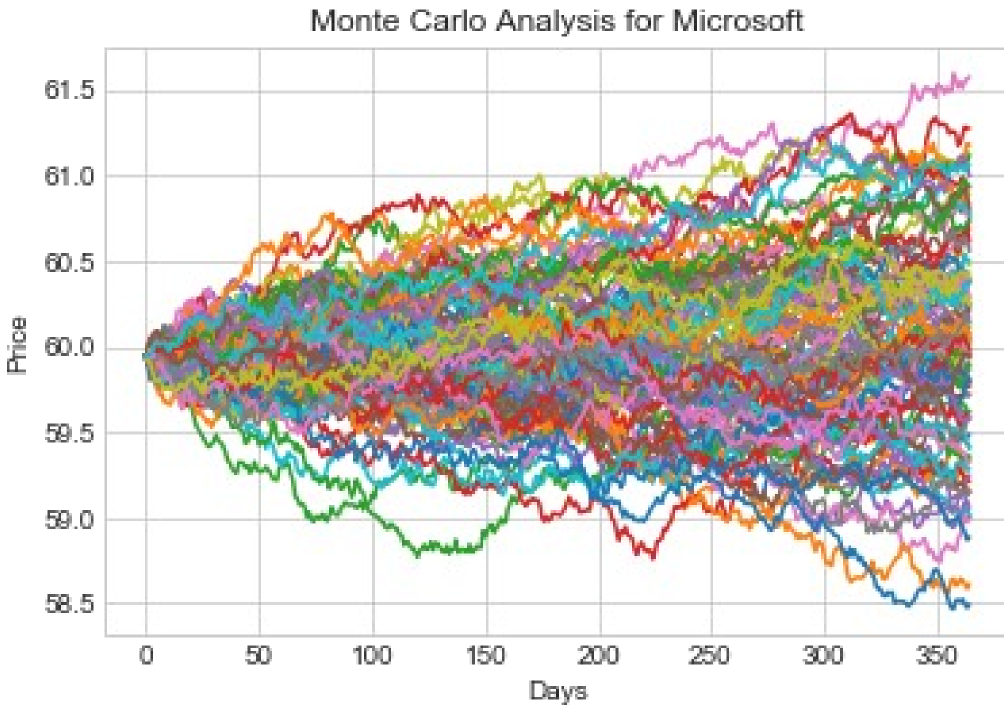
	Open	High	Low	Close	Volume	Daily Return
Date						
2016-10-24	117.10	117.74	117.00	117.65	23538673	NaN
2016-10-25	117.95	118.36	117.31	118.25	48128970	0.005100
2016-10-26	114.31	115.70	113.31	115.59	66134219	-0.022495
2016-10-27	115.39	115.86	114.10	114.48	34562045	-0.009603
2016-10-28	113.87	115.21	113.45	113.72	37861662	-0.006639

```
start_price = 117.10

for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))

plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Apple')
```

```
<matplotlib.text.Text at 0x12993d29630>
```




```
# For Microsoft Stock - MSFT
MSFT.head()

                Open   High    Low  Close    Volume
Date
2016-10-24  59.94  61.00  59.93  61.00  54066978
2016-10-25  60.85  61.37  60.80  60.99  35137164
2016-10-26  60.81  61.20  60.47  60.63  29911608
2016-10-27  60.61  60.83  60.09  60.10  28479856
2016-10-28  60.01  60.52  59.58  59.87  33574684

start_price = 59.94

for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))

plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Microsoft')

<matplotlib.text.Text at 0x12993e8c198>
```



Let's go ahead and get a histogram of the end results for a much larger run. (note: This could take a little while to run , depending on the number of runs chosen)

```
# Lets start with Google stock price
start_price = 830.09

# Set a large numebr of runs
runs = 10000

# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)

for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)[days-1]
```

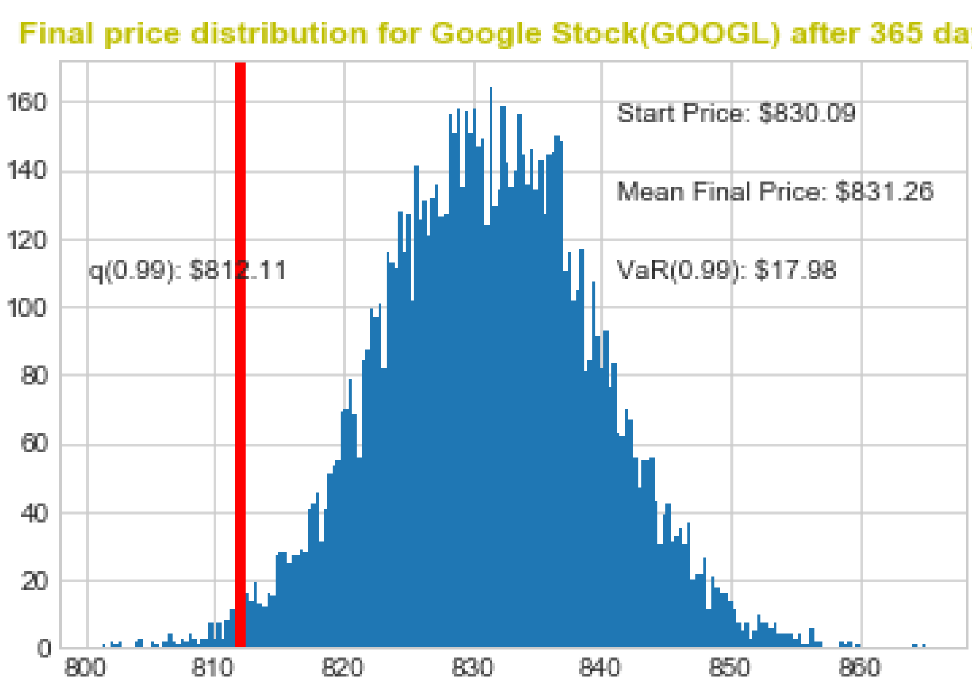
Now that we have our array of simulations, we can go ahead and plot a histogram ,as well as use qunatile to define our risk for this stock.

For more info on quantiles, check out this link: http://en.wikipedia.org/wiki/Quantile

```
# Now we'll define q as the 1% empirical quantile, this basically means that 99% of the values should fall between here
q = np.percentile(simulations,1)
```

```
# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')

# For plot title
plt.title(s="Final price distribution for Google Stock(GOOGL) after %s
days" % days, weight='bold', color='Y')
```

```
<matplotlib.text.Text at 0x12999daf4a8>
```



Awesome! Now we have looked at the 1% empirical quantile of the final price distribution to estimate the Value at Risk for the Google Stock(GOOGL), which looks to be $17.98 for every investment of 830.09 (The price of one initial Google Stock).

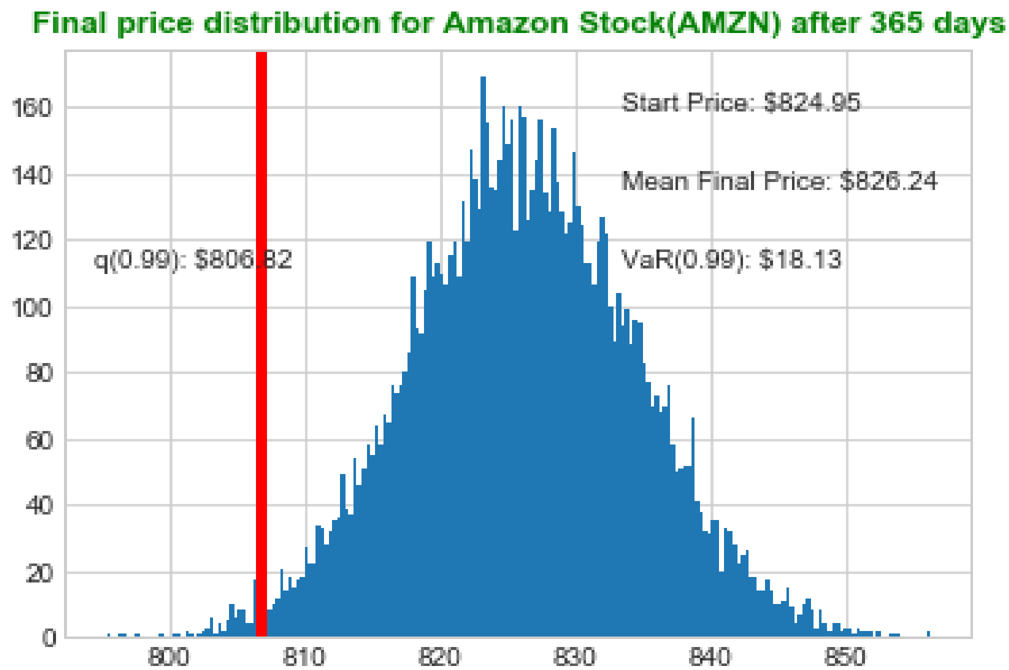
This basically means for every initial GOOGL stock you purchase you're putting about $17.98 at risk 99% of the time from our Monte Carlo Simulation.

Now lets plot remaining Stocks to estimate the VaR with our Monte Carlo Simulation.

```
# For Amazon Stock Price
start_price = 824.95

# Set a large numebr of runs
runs = 10000

# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)

for run in range(runs):
    # Set the simulation data point as the last stock price for that 
run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]


# Now we'll define q as the 1% empirical quantile, this basically 
means that 99% of the values should fall between here
q = np.percentile(simulations,1)

# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)

# Using plt.figtext to fill in some additional information onto the 
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')

# For plot title
```

```
plt.title(s="Final price distribution for Amazon Stock(AMZN) after %s
days" % days, weight='bold', color='G')
```

```
<matplotlib.text.Text at 0x1299a123630>
```

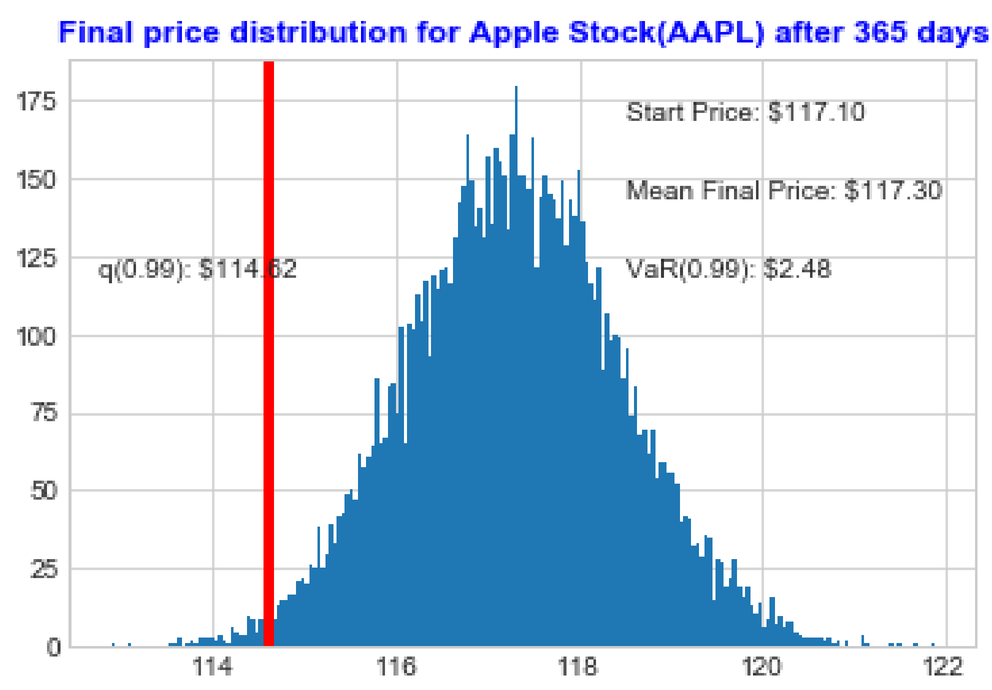


This basically means for every initial AMZN stock you purchase you're putting about $18.13 at risk 99% of the time from our Monte Carlo Simulation.

```
# For Apple Stock Price
start_price = 117.10

# Set a large numebr of runs
runs = 10000

# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)

for run in range(runs):
    # Set the simulation data point as the last stock price for that
run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]


# Now we'll define q as the 1% empirical quantile, this basically
means that 99% of the values should fall between here
q = np.percentile(simulations,1)

# Now let's plot the distribution of the end prices
```

```
plt.hist(simulations, bins=200)

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')

# For plot title
plt.title(s="Final price distribution for Apple Stock(AAPL) after %s
days" % days, weight='bold', color='B')
```

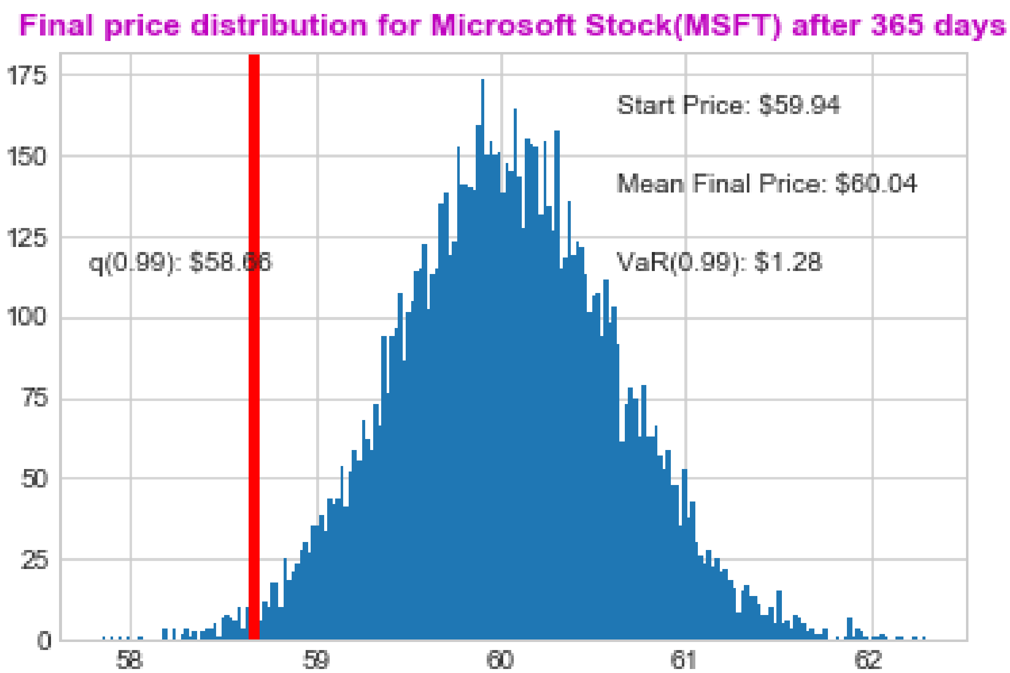
```
<matplotlib.text.Text at 0x1299a2a0ba8>
```



Great! This basically means for every initial AAPL stock you purchase you're putting about $2.48 at risk 99% of the time from our Monte Carlo Simulation.

```python
# For Microsoft Stock Price
start_price = 59.94

# Set a large numebr of runs
runs = 10000

# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)

for run in range(runs):
    # Set the simulation data point as the last stock price for that
run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]

# Now we'll define q as the 1% empirical quantile, this basically
means that 99% of the values should fall between here
q = np.percentile(simulations,1)

# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')

# For plot title
plt.title(s="Final price distribution for Microsoft Stock(MSFT) after
%s days" % days, weight='bold', color='M')
```

```
<matplotlib.text.Text at 0x1299ab54860>
```



Nice, This basically means for every initial MSFT stock you purchase you're putting about $1.28 at risk 99% of the time from our Monte Carlo Simulation.

Now lets estiamte the Value at Risk(VaR) for a stock related to other domains.

We'll estimate the VaR for:

- Johnson & Johnson > JNJ (U.S.: NYSE) JNJ
- Wal-Mart Stores Inc. > WMT (U.S.: NYSE) WMT
- Nike Inc. > NKE (U.S.: NYSE) NKE

By using the above methods to get Value at Risk.

```
# List of NYSE_stocks for analytics
NYSE_list = ['JNJ','NKE','WMT']

# set up Start and End time for data grab
end = datetime.now()
start = datetime(end.year-1,end.month,end.day)

#For-loop for grabing google finance data and setting as a dataframe
# Set DataFrame as the Stock Ticker

for stock in NYSE_list:
    globals()[stock] = DataReader(stock,'google',start,end)
```

Let's go ahead and play aorund with the JNJ(Johnson & Johnson) Stock DataFrame to get a feel for the data.

```
JNJ.head()
```

```
                  Open    High     Low   Close    Volume
Date
2016-11-03  114.88  115.44  114.75  115.03   6227110
2016-11-04  115.04  115.94  115.04  115.11   7160570
2016-11-07  115.89  116.72  115.84  116.66   6398177
2016-11-08  116.48  117.57  116.47  117.05   6675268
2016-11-09  120.00  122.50  118.10  120.31  16219977
```

```
JNJ.describe()
```

```
             Open        High         Low       Close        Volume
count  249.000000  250.00000  249.000000  250.00000  2.500000e+02
mean   125.147831  125.89244  124.554900  125.28044  6.244547e+06
std      8.173767    8.27359    8.151047    8.18848  2.276207e+06
min    110.540000  111.20000  109.320000  110.99000  1.023583e+06
25%    116.120000  116.69750  115.760000  116.30500  4.731960e+06
50%    125.620000  126.42500  125.130000  125.92500  5.739157e+06
75%    132.280000  133.15500  131.620000  132.43000  7.300170e+06
max    143.380000  144.35000  142.080000  143.62000  1.621998e+07
```

```
JNJ.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 250 entries, 2016-10-28 to 2017-10-25
Data columns (total 5 columns):
Open      249 non-null float64
High      250 non-null float64
Low       249 non-null float64
Close     250 non-null float64
Volume    250 non-null int64
dtypes: float64(4), int64(1)
memory usage: 11.7 KB
```

Now that we've seen the DataFrame, let's go ahead and plot out the closing prices of NYSE stocks.

```
# Let's see a historical view of the closing price for JNJ(Johnson & Johnson)
JNJ['Close'].plot(title='Closing Price - JNJ',legend=True, figsize=(10,4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cb33fe19b0>
```


```
# Let's see a historical view of the closing price for NKE(Nike Inc.)
NKE['Close'].plot(title='Closing Price - NKE',legend=True,
figsize=(10,4))

<matplotlib.axes._subplots.AxesSubplot at 0x1cb33f3fc18>
```


```
# Let's see a historical view of the closing price for WMT(Wal-Mart
Stores Inc.)
WMT['Close'].plot(title='Closing Price - WMT',legend=True,
figsize=(10,4))

<matplotlib.axes._subplots.AxesSubplot at 0x1cb34059a58>
```

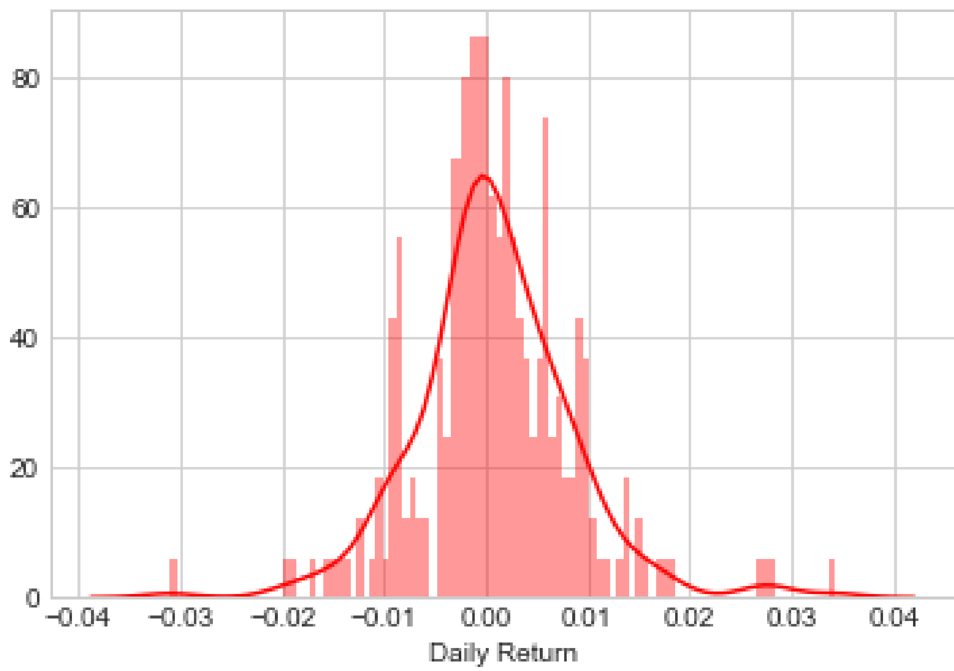
Value at risk using the "Bootstrap" method

we will calculate the empirical quantiles from a histogram of daily returns.

Let's go ahead and use pandas to retrieve the daily returns for the JNJ, WMT & NKE stock.

```
# We'll use pct_change to find the percent change for each day

#For JNJ stocks
JNJ['Daily Return'] = JNJ['Close'].pct_change()

# Note the use of dropna() here, otherwise the NaN values can't be
read by seaborn
sns.distplot(JNJ['Daily Return'].dropna(),bins=100,color='R')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1df64345668>
```



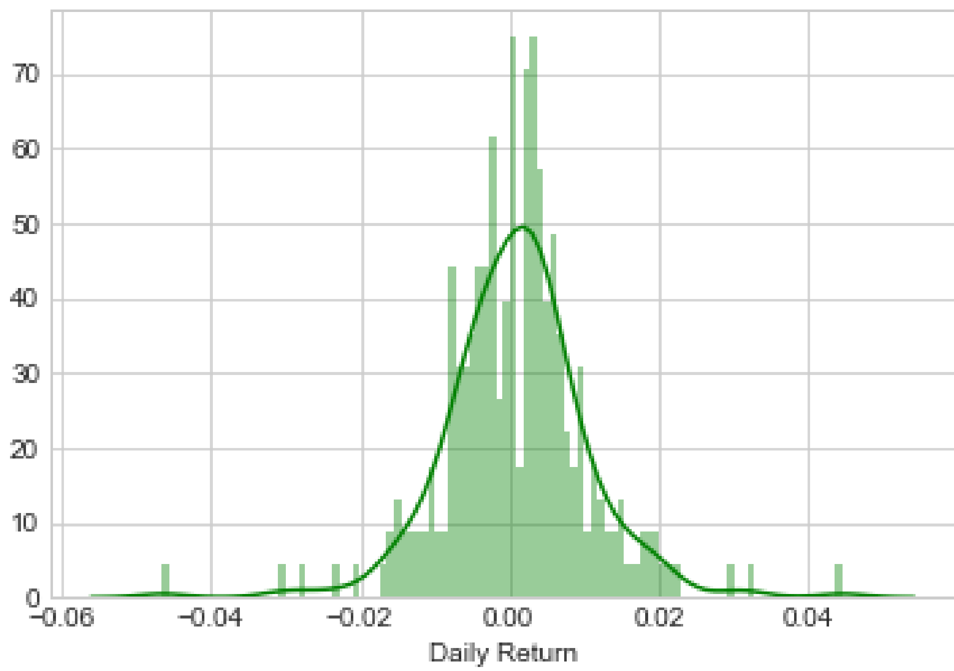

```
(JNJ['Daily Return'].dropna()).quantile(0.05)
```

```
-0.010277273373901852
```

The 0.05 empirical quantile of JNJ stock daily returns is at -0.010. That means that with 95% confidence, our worst daily loss will not exceed 1%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.010 * 1,000,000 = $10,000.

```
# For WMT stocks
WMT['Daily Return'] = WMT['Close'].pct_change()

sns.distplot(WMT['Daily Return'].dropna(),bins=100,color='G')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1df65aaf470>
```



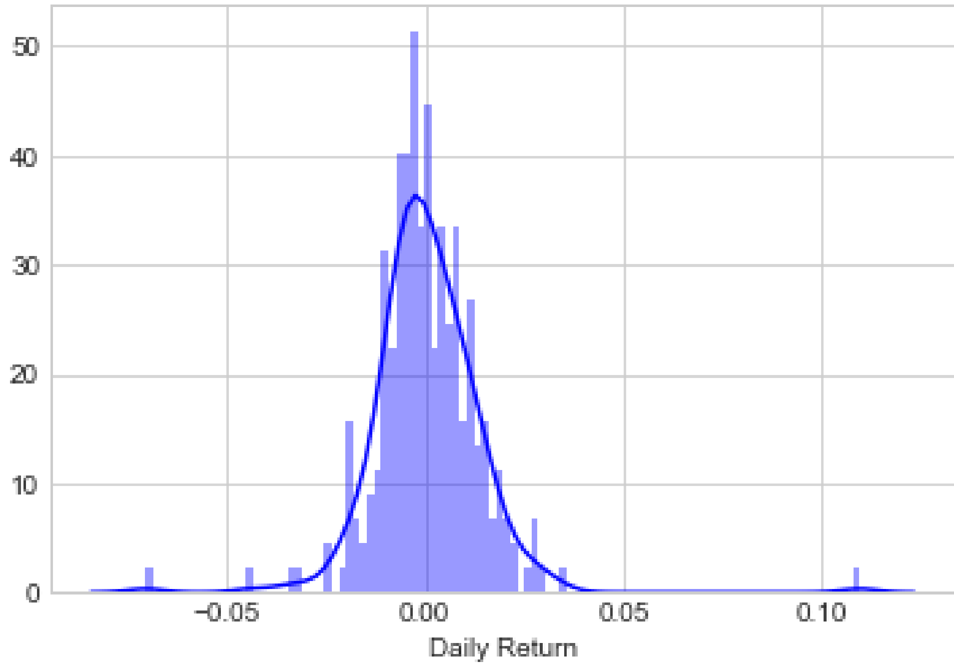

```
(WMT['Daily Return'].dropna()).quantile(0.05)
```

```
-0.013540544480876137
```

The 0.05 empirical quantile of WMT stock daily returns is at -0.013. That means that with 95% confidence, our worst daily loss will not exceed 1.3%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.013 * 1,000,000 = $13,000.

```
# For NKE stocks
NKE['Daily Return'] = NKE['Close'].pct_change()

sns.distplot(NKE['Daily Return'].dropna(),bins=100,color='B')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1df6589e908>
```




```
(NKE['Daily Return'].dropna()).quantile(0.05)
```

```
-0.018434055794082728
```

The 0.05 empirical quantile of NKE stock daily returns is at -0.018. That means that with 95% confidence, our worst daily loss will not exceed 1.8%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.018 * 1,000,000 = $18,000.